

United States Patent Application

of

Adrian E. Colley

Peter C. Jones

Robert W. Scheifler

Michael P. Warres

and

Ann Wollrath

for

**Method and System for Passing Objects in
a Distributed System using Serialization Contexts**

LAW OFFICES

FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

0075366-040404

FIELD OF THE INVENTION

The present invention relates generally to data processing systems and, more particularly, to passing serialized versions of objects in a distributed system.

BACKGROUND OF THE INVENTION

Distributed systems can be made up of various components, including both hardware and software. A distributed system (1) allows its users to share services and resources over a network of many devices; (2) provides programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplifies the task of administering the distributed system.

A distributed system can be implemented using an object-oriented programming language, such as Java™. The Java™ programming language is typically compiled into a platform-independent format, using a bytecode instruction set, which can be executed on any platform supporting the Java™ virtual machine. The Java™ programming language is described in greater detail in The Java™ Language Specification by James Gosling, Bill Joy, and Guy Steele, Addison-Wesley, 1996, which is incorporated herein by reference. Java™ and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Distributed systems require that programs running in different address spaces be able to communicate with each other. In a system using an object-

oriented programming language, such as the Java™ programming language, this communication can be achieved by passing an "object," which represents an item or instance manipulated by the system, from one program to another. In such a system, a "class" provides a template for the creation of objects having characteristics of that class. The objects in each class share certain characteristics or attributes determined by the class. A class thus defines the type of an object. Objects are typically created dynamically during system operation. Methods associated with a class are generally invoked on the objects of the same class or subclass.

In a Java™ distributed system, an object is referred to as being remote when its methods can be invoked from another address space, typically a Java™ virtual machine on a different computer. A remote object is described by one or more remote interfaces, which are Java™ interfaces that declare the methods of the remote object. Remote Method Invocation (RMI) is used to invoke a method of a remote interface on a remote object. RMI is explained in, for example, the Remote Method Invocation Specification, Sun Microsystems, Inc. (1997) available at

<http://www.java.sun.com/products/jdk/1.2/docs/guide/rmi/spec/rmiTOC.doc.html>, which is incorporated herein by reference.

As part of RMI, Java™ objects are passed between a client and a server. Before being passed, a Java™ object is converted into a serialized representation

5

10

15

20

LAW OFFICES

FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000

10

15

20

LAW OFFICES

5 A method consistent with the present invention passes a first object and a second object, both instances of a class, in distinct remote method calls in a distributed system. The first object is passed from a sender to a recipient with a descriptor of the class and a handle corresponding to the descriptor. The handle and the descriptor are stored by the recipient. The second object is then passed from the sender to the recipient with the handle, and the recipient uses the handle to determine the descriptor.

10 BRIEF DESCRIPTION OF THE DRAWINGS

This invention is pointed out with particularity in the appended claims. The above and further advantages of this invention may be better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

15 Fig. 1 depicts a distributed system 100 suitable for practicing methods and systems consistent with the present invention;

Fig. 2 is a block diagram showing two serialization contexts consistent with the present invention;

Fig. 3 depicts a flow chart of a method for passing objects using serialization contexts, consistent with the present invention;

20 Fig. 4 is a flow chart of the "handshake" between a sender and a recipient to agree on a serialization context pair to use; and

Fig. 5 is a flow chart showing how the committed flag can be used to provide two-way communication.

DETAILED DESCRIPTION

A system consistent with the present invention reduces the number of redundant class descriptors that are sent during remote method calls by using serialization contexts. "Serialization contexts" are dictionary objects that map a class descriptor to a corresponding integer handle. When possible, the integer handle, rather than the full class descriptor, is passed, saving processing time in RMI calls.

Figure 1 depicts a distributed system 100 suitable for practicing methods and systems consistent with the present invention. Distributed system 100 includes client computer 102 and server computer 104, communicating via network 106. Network 106 may be, for example, a local area network, wide area network, or the Internet.

Client computer 102 includes a memory 108, a secondary storage device 110, a central processing unit (CPU) 112, an input device 114, and a video display 116. The memory 108 includes a Java™ runtime system 118. The Java™ runtime system 118 includes a Java™ virtual machine 120, and a Java™ remote method invocation (RMI) system 122. The RMI system 122 contains one

or more serialization contexts 124. Memory 108 also includes a program 126 running on client computer 102.

Server computer 104 includes a memory 128, a secondary storage device 130, a central processing unit (CPU) 132, an input device 134, and a video display 136. The memory 128 includes a Java™ runtime system 138. The Java™ runtime system 138 includes a Java™ virtual machine 140, and the Java™ remote method invocation (RMI) system 142. The RMI system 142 contains one or more serialization contexts 144. Memory 128 also includes a program 146 running on server computer 104, and one or more objects 148.

Using RMI, objects can be passed between client computer 102 and server computer 104. For example, a program 146 running on client computer 102 can invoke a method on an object 148 stored in the memory 130 of server computer 104. Client computer 102 would use RMI system 122 to convert the method call, including an identification of the remote method and any parameters, into a byte stream that is sent to server computer 104 via network 106. Server computer 104, upon receiving the byte stream, would use its RMI system to convert the byte stream into executable bytecode and initiate the invocation of the method on the remote object. If the method results in a return value, server computer 104 would convert the return value to a byte stream using its RMI system, and transmit the byte stream to the client computer 102.

06502.0267

The byte streams contain serialized versions of Java™ objects, e.g. parameters or return values. A serialized object contains two main parts: the object data and a class descriptor. The class descriptor describes the content and format of the object data. Object serialization is explained in, for example, the Java™ Object Serialization Specification, available at <http://java.sun.com/products/jdk/1.3/docs/guide/serialization/spec/serialTOC.doc.html>, which is incorporated herein by reference.

Within a single remote method call, a class descriptor is sent with the first object of that type that is serialized, subsequent objects of that type in the same remote method call refer to the class descriptor with a "back reference" (i.e., an integer handle).

Serialization Contexts

In a serialized object, the class descriptor provides the full name of the class and its serialization ID, which uniquely identifies the class. The serialization ID is a 64-bit hash of the class name, interface class names, methods, and fields. Each class descriptor is an instance of the Java™ class ObjectOutputStreamClass, defined as follows:

```
public class ObjectOutputStreamClass
{
    public static ObjectOutputStream lookup(Class cl);
    public String getName();
    public Class forClass();
    public ObjectOutputStreamField[] getFields();
    public long getSerialVersionUID();
    public String toString();
}
```


},

Serialization contexts can be used to pass the class descriptors of serialized objects. As explained above, serialization contexts are dictionary objects that map a class descriptor to a corresponding integer handle. When possible, the integer handle, rather than the full class descriptor, is passed, saving processing time in RMI calls.

Figure 2 is a block diagram showing serialization contexts 124 and 144 in more detail, consistent with the present invention. Each serialization context is maintained as a pair of tables: one for outgoing handles, e.g., 202 or 206, and one for incoming handles, e.g., 204 or 208. Outgoing handles are used when a program running on the computer acts as a sender (e.g., makes a remote call or sends return values). Incoming handles are used when a program running on the computer acts as a recipient (e.g., receives a remote call or receives return values). In this way, a program 126 running on the client computer and a program 146 running on the server computer can each act as a sender or recipient. Both the RMI system of the client computer and the RMI system of the server computer maintain an outgoing handle table and an incoming handle table.

RMI system 122 of client computer 102 contains serialization context 124, which consists of outgoing handle table 202 and incoming handle table 204, and RMI system 142 of server computer 104 contains serialization context 144, which consists of outgoing handle table 206 and incoming handle table 208. Each incoming handle table has one or more entries including a handle and a class

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995

10

15

20

5 true, as detailed with reference to one embodiment in Figure 5 below, the sender retrieves the handle corresponding to the class descriptor from the outgoing handle table 202 of serialization context 124, and sends the handle rather than the full class descriptor to the recipient (step 306). The recipient then uses the handle to look up the class descriptor in the incoming handle table 208 of serialization context 144. If the class descriptor that the sender wishes to send is not in the outgoing handle table 202 of serialization context 124, the sender sends both the class descriptor and a new handle (step 310). For subsequent calls, the sender can send just the handle to the recipient.

Handshake

10 Figure 4 is a flow chart of the "handshake" between a sender and a recipient to agree on a serialization context pair to use. When a connection between the sender and the recipient is established, e.g., when a new RMI session begins, the sender and recipient "handshake" to agree on a serialization context pair to use, as stated in step 302 of figure 3 above. Each pair of serialization contexts, e.g., serialization contexts 124 and 144, is identified by a globally unique context ID. This context ID is used to perform the handshake. First, the sender determines whether one of the sender's serialization contexts is associated with a serialization context of the recipient (step 402). If so, the sender transmits the context ID for that serialization context pair to the recipient (step 404). Otherwise, the sender transmits a null ID to the recipient (step 406). If the recipient receives a non-null context ID (step 408), it checks to see if it still has the corresponding serialization

10

15

20

DIS

- 12 -

5

10

15

20

DIS

As part of this process, the recipient enters any new handle/class descriptor pairs into the incoming handle table 208. In one embodiment, this updating occurs before the method call can return successfully to the client. Therefore, when the remote method call is successfully returned to the original sender, the handle/class descriptor pair is implicitly acknowledged, and the sender can set the corresponding committed flag in the outgoing handle table 202 to true.

Handle acknowledgment - return values

Class descriptors used by the recipient (e.g., program 126 running on server computer 104) to send return values to the sender (e.g., program 146 running on client computer 102) require an explicit acknowledgment. The recipient has no way of knowing whether the sender successfully stored the handle/class descriptor pair sent with the return value in the incoming handle table 204. To acknowledge that the incoming handle table 204 has been updated, the sender sends an explicit acknowledgment of its successful receipt of the handle/class descriptor pair with its next call to the recipient. The acknowledgment can be delayed in this way because the recipient only needs the acknowledgment if there are future communications between the sender and the recipient.

Garbage collection

Serialization contexts can get quite large. If a pair of tables is no longer needed, memory space can be saved by deleting the tables. Preferably, this "garbage collection" is made possible by using the globally unique ID codes corresponding to each serialization context pair. A table that maps serialization

contexts to their unique ID codes can be maintained by, for example, RMI system 124 or RMI system 144. Space in this table is "leased," meaning that after a set amount of time has passed, a serialization context/unique ID code pairing is deleted from the table. Each time a serialization context is accessed by an object, e.g., a program running on client computer 102 or server computer 104, the lease time is reset. Therefore, serialization contexts will automatically be available for a set amount of time between uses. After the set amount of time expires and a serialization context is deleted, a new table is created when a client wishes to communicate with the server, as described in Figure 5.

Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and examples be considered as exemplary only, with a true scope and spirit of the invention being indicated by the following claims.

10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995

LAW OFFICES

FINNEGAN, HENDERSON,
FARABOW, GARRETT,
& DUNNER, L.L.P.
1300 I STREET, N. W.
WASHINGTON, DC 20005
202-408-4000